FINE-GRAINED INTEGRATED MANAGEMENT OF SOFTWARE CONFIGURATIONS AND TRACEABILITY RELATIONS

Pietro Colombo[1], Vieri del Bianco[1] and Luigi Lavazza[1,2]
[1]*Dipartimento di Informatica e Comunicazione, Università dell'Insubria, Via Mazzini, 5 – 21100 Varese, Italy*
[2]*CEFRIEL, Via Fucini 2 – 20133 Milano, Italy*

Keywords: Model versioning, XML.

Abstract: Software Configuration Management is essential to manage the evolution of non trivial software systems. Requirements for SCM support are continuously growing, demanding for seamless integration with the traditional development tools and support for management activities like change management, change impact analysis, etc. This paper presents SCCS-XP, a SCM platform supporting change management and traceability among fine-grained software artifacts. SCCS-XP exploits a XML-based language to represent versioned software artifacts. SCCS-XP provides the basic capabilities to build full-fledged SCM environments featuring traceability management, change management and integrates nicely with development tools.

1 INTRODUCTION

Software Configuration Management (SCM) is the discipline of managing the evolution of large and complex software systems (Tichy, 1988). SCM provides both functionalities that assist developers in performing coordinated changes to software products and change management (CM) services, like impact analysis. In the case of impact analysis the traceability relations are used to determine the consequences of changes in requirements or in other software artifacts. The level of support to the configuration and change management activities can be classified with respect to the following issues:

- *Abstraction*. Tools should be as orthogonal as possible to the structure and contents of the managed items, as well as to the relations involving items.
- *Granularity*. The platform has to be able to deal with composition hierarchies. Since we do not know a priori the requirements for the SCM and CM system, the platform should treat in a uniform way all the elements of the hierarchy.
- *Automation*. In order to be effective, SCM activities must be largely supported by automated functionalities. Besides, since software development typically exploits a variety of development tools, it is necessary that the SCM system exchanges data seamlessly with such tools.

The proposed platform aims at supporting version and configuration management, as well as the typical change management activities. We do not intend to provide a full-fledged SCM and/or change management environment, but a platform with the functionalities needed to build such environment.

We identified the following requirements for the platform:

- Flexibility. The platform has to support the building of different functionalities, strategies, and policies. No constraint is expressed on the model, methodology or process; different tool builders can build their own tool by implementing the features needed in each specific environment.
- The platform has to be relatively light-weight. It has to exploit an existing DBMS to focus on the provision of functionalities specifically oriented to SCM and CM. Moreover, it has to be based on a standard DBMS API, in order to exploit any available implementation, thus increasing portability and interoperability.
- The platform has to support a development environment where SCM and change management functions are effectively integrated with the development tools.

It can be noted that several of the properties mentioned above call for a representation of artifacts that is standard, precise, detailed, and supported by data management systems. From this point of view

Colombo P., del Bianco V. and Lavazza L. (2008).
FINE-GRAINED INTEGRATED MANAGEMENT OF SOFTWARE CONFIGURATIONS AND TRACEABILITY RELATIONS.
In *Proceedings of the Third International Conference on Software and Data Technologies - SE/GSDCA/MUSE*, pages 159-164
DOI: 10.5220/0001883501590164

the technology is mature to build the desired SCM platform. XML perfectly fits the requirements since it enables the representation of information at the required granularity level. XML was chosen both to represent and to exchange data with the development tools. Our platform is thus called SCCS-XP (Software Change and Configuration System XML Platform).

The paper is organized as follows: Section 2 presents the conceptual organization and implementation of SCCS-XP. Section 3 presents a set of case studies where SCCS-XP is used to build SCM tools. Section 4 accounts for related work. Section 5 presents some conclusions and illustrates future work.

2 SCCS-XP: CONCEPTS AND IMPLEMENTATION

In this section we define the architecture of SCCS-XP.

2.1 Conceptual Organization

The proposed architecture is conceptually organized in an *Abstract layer*, featuring an *Abstract data model*, a *Uniform layer*, featuring the *Uniform data model*, and a *Model layer*, featuring *Specific data models* (see Figure 1).

Conceptually the *Abstract layer* is simple, as it corresponds to a DBMS featuring a standard query language: the *Abstract data model* corresponds to the logic data model of the adopted DBMS. The functions provided by the *Abstract layer* do not make distinction between different types of artifacts.

A relevant requirement for the abstract data layer is that it must be easy to represent the data belonging to the *Uniform layer* by means of the *Abstract data model*. Another requirement is that it has to provide some sort of aggregation mechanism, in order to represent data at the suitable granularity level.

The *Uniform layer* supports the evolution process, with particular reference to versioning, traceability and change management.

The *Uniform data model* defines the concept of artifact on the basis of the *Abstract data model*; in particular it exploits the aggregation mechanism to provide access at different granularity levels. The *Uniform layer* can then be regarded as organized internally in two sub-layers: the first one exploits the *Abstract layer* to make artifacts versionable, while the second sub-layer provides functionalities to support traceability and change management, which operate on versioned artifacts (or on their parts).

We decided to build the Versioned DBMS (VDBMS) exploiting a XML-based DBMS. The language employed to internally represent versioned artifacts is based on XML (see Section 2.3). Since the activities carried out at the upper layers are typically supported by existing tools it is essential that they interface with the platform: SCCS-XP supports loose integration at the data level by means of dynamically generated Virtual Files (Leblang, 1994). Translators have to be developed to convert the tools' specific formats into the format supported by SCCS-XP. Virtual Files do not exist as files in the SCCS-XP platform: actual files are produced upon request by composing the information available in the platform. This guarantees openness and scalability of the platform.

The *Specific layer* includes the tools that are dedicated to the crafting of the different kinds of artifacts that are created and managed throughout the software lifecycle. These tools employ different data models to represent the data that have to be exchanged with the kernel of the platform.

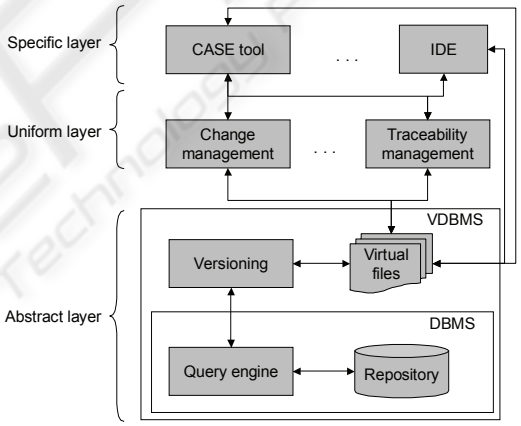

Figure 1: SCCS-XP Conceptual Architecture.

The organization of SCCS-XP requires that the data managed by the external tools can be converted (through Virtual Files) into/from data conformant to the *Uniform data model*. The conversion must be possible for every data format supported by the tools and subject to versioning, and involved in traceability and control management. When XML is not directly supported, translators can be used.

2.2 The Versioning Model

The versioning model supported by SCCS-XP is traditional. According to the classification proposed by Conradi and Westfechtel (1998), our model features symmetric deltas, and directly implements extensional versioning. Intensional versioning can

also be achieved, by means of the querying capabilities of the Versioned DBMS.

As already mentioned, SCCS-XP aims at managing XML artifacts. Since an XML tag could represent a versioned item, we need to version single tags. This is achieved by coupling the product model and the versioning model.

Custom relations can be defined by means of suitable plug-ins (see Section 2.4). Specialized relations among versioned artifacts are defined by means of XML elements belonging to the *Uniform data model*.

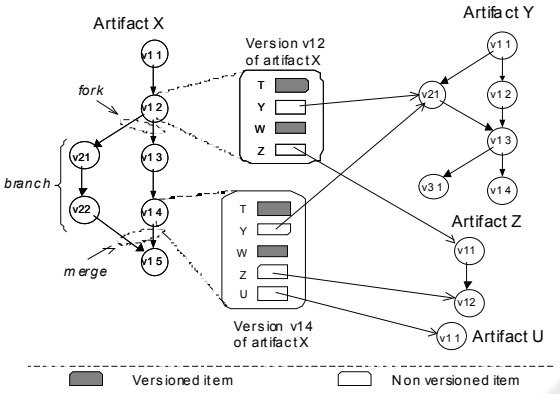

Figure 2: Artifact composition and fine-grained versioning.

2.3 Internals of the Versioned DBMS

In order to make the implementation of the platform practical, we needed to make the data models as uniform as possible. For this purpose we defined a meta-model that can be used to define the various data models used in the platform. In this way, all the data models share a common layer; moreover their definition, correspondences, transformations, etc. are precisely defined in the meta-model. In particular, the meta-model has to support data definition, data exchange procedures, data persistency and querying. XML provides the most comprehensive support for the aforementioned features, thus it was employed for the *Abstract layer*.

Having adopted XML for the *Abstract layer*, we had to make XML data versionable. For this purpose, we defined a XML-based language named VerML (Versioned XML), which supports the representation of the versioning information contained in the models illustrated in Section 2.2.

2.4 Implementation

The implementation of the *Abstract layer* of SCCS-XP is based on an XML DBMS. Versioning of XML tags was implemented according to the models presented in Section 2.2. For this purpose we adopted the following strategy:

- XML data are stored exploiting the functionality of the DBMS.
- It is possible to decompose a piece of XML data into individual XML tags that are stored separately, thus implementing a fine-grained repository.
- Information concerning the composition of documents is contained in VerML data, which is stored in the DBMS as well.
- Different versions of the same artifact, possibly belonging to a composite artifact, are individually stored.
- Versioning information is also stored in VerML data. For instance, consider the following situation: class C contains attribute A and method M, and there are two versions of method M. The database contains artifacts representing attribute A, version 1 of method M, version 2 of M, version 1 of C and version 2 of C. Each version of C contains a VerML tag pointing to the correct version of M.

The *Abstract layer* of SCCS-XP implements the operations of the VDBMS (see Figure 1): creation of a new versioned artifact, insertion of a new version, extraction of a given version of a (possibly composed) artifact, etc. Basic check-in and check-out services were implemented as part of the *Abstract layer*. They are accessible via a simple command line interface. The operations can involve any XML tag individually stored; therefore it is possible to extract pieces of a document. In this way the *Abstract layer* provides the *Uniform layer* with the flexibility needed to build the *Uniform data model* required by the user.

Locking is currently not implemented; SCCS-XP does not manage safe concurrent access to the versioned data.

Items stored in the database are manipulated by means of queries. The querying capabilities were implemented using XPath (Clark and DeRose, 1999); the more powerful XQuery (Boag et al., 2007) will be considered for future improvements of SCCS-XP XML query engine. The implementation of the Uniform layer is based on the modular plug-in architecture described in Figure 3.

The *Uniform data model* is obtained by means of suitable plug-ins that define data and operations on the basis of the abstract data layer. For instance, we developed the Java plug-in, which is able to parse Java files and translate them into XML data, which can then be versioned and stored by means of the services provided by the *Abstract layer*. The Java

plug-in can also retrieve a given version of a Java XML artifact and un-parse it, generating a Java text file that can be read by any suitable development tool. In order to manage requirements expressed in natural language, we developed a plug-in, named SimpleReq, which parses LaTeX files and builds a XML hierarchical representation of the requirements contained in LaTeX files.

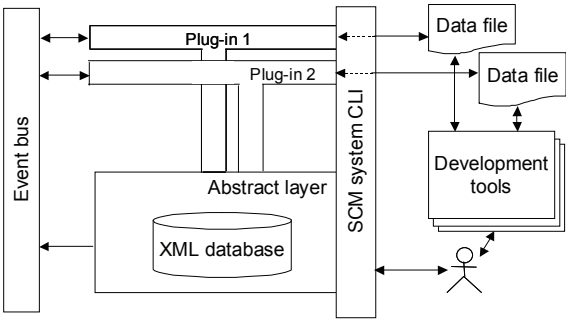

Figure 3: Architecture of SCCS-XP based SCM tools.

A difficulty in managing data exchange with external tools consists in understanding which parts of a given document correspond to elements already stored in the platform and which parts are new, thus requiring the creation of new sub-element versions. For this purpose it is possible to insert annotations in the files exported from SCCS-XP. For instance the Java plug-in exports a source code file containing annotations, so that when editing it, the user can recognize the annotations and preserves them; when the plug-in re-imports the modified file, it can recognize annotations as marking parts corresponding to existing artifacts. A plug-in could also exploit the XML diff techniques (Mouat, 2002).

Besides managing the conversion of data between the format understood by external tools and the format supported by the *Abstract layer*, the plug-in can be employed for more complex goals. The StructureX plug-in introduces new artifact types, and uses them to correlate the versioned artifacts with an external hierarchical file system (see Section 3). The new artifacts representing files, directories, etc. are managed by the *Abstract layer* in the usual way, but are not made visible to the user, who interacts with these objects only through a suitable interface.

The more complex plug-in named RelationX (see Section 3) reacts to artifact changes in order to maintain referential integrity. For this purpose the plug-in is connected to the event bus: it receives the relevant events from the *Abstract layer* and triggers operations that update the links between artifacts that are defined by the plug-in itself.

Every plug-in can add or modify the commands provided by the command line interface (CLI) originally defined by the *Abstract layer*. During the usage of the system, the user can declare a context corresponding to a plug-in: this determines which commands are executed in case of conflicts.

3 VALIDATION

In order to assess the validity of SCCS-XP we employed it in three different applications, each conceived to exercise a specific set of features of the platform.

The first application consisted in the development of a basic single user SCM tool “a la CVS” (XVS). The goal was to assess the ability of SCCS-XP with respect to effective and efficient management of data of realistic size. XVS features the usual functionality: check-in, check-out, branching, etc. XVS uses the Java plug-in to version fragments of source code; in particular, class and methods are versioned as separate artifacts.

SCCS-XP was then employed to develop a second application, with the purpose of showing that the platform can be used to build a proper configuration management tool. The main goal of this tool, called XCM, was to provide the user with the possibility of managing configurations consisting of coherent sets of file versions. For this purpose the concept of structured project was introduced. Such concept is implemented by means of a new artifact, which is treated by the *Abstract layer* as any other artifact. We developed a specific plug-in (called StructureX) to deal with the specificity of the concept of project: new versions of a project are created whenever files or directories are added, changed or removed from the project. StructureX interprets project artifacts and correlates them to the other artifacts and the underlying file system. Thanks to StructureX, the concept of artifact evolves independently from the hierarchical structure (the file system) which it belongs to. StructureX can be reused in different contexts, for instance it can be used to enhance XVS, providing the user with the possibility of versioning both Java files and projects containing such files. In fact, we actually added StructureX to XVS (obtaining XCM). Consider that StructureX employs the *Abstract layer* in order to define an enhancement of the *Uniform layer* data model, in which the concepts of project, directory and files are defined (see Figure 1).

Finally, a third plug-in named RelationX, was developed to manage the semantic relations among artifacts. In particular, we considered the typical traceability relations that link requirements specification documents (managed by the SimpleReq plug-in) to the source code, and the relations that link the code with unit test cases. The

goal of the application was to perform impact analysis on the basis of the known relations among artifacts. The application handles three types of relations:

- Relations already present in the data managed at the level of the *Abstract layer*. For instance, the composition relationships and the derivation and branch relationship.
- Relations that have to be specified explicitly by the user: for instance, the relations that link a requirement to the classes that implement it. These relations are represented extensionally.
- Finally, some relations can be inferred from the information normally present in the repository. For instance, the dependence of a method M from a given requirement R may be inferred by the fact that M belongs to class C, and class C implements R. In order to ease the management of relations, they are expressed intensionally by rules that the plug-in is able to interpret, thus freeing the user from explicitly entering them.

The RelationX plug-in enhances the *Uniform layer* data model with the aforementioned types of relations. This plug-in is able, given a (versioned) artifact, to exploit the knowledge of the relations for finding the set of related (versioned) artifacts. This ability can be used to reason about dependencies among artifacts.

4 RELATED WORK

The Software Configuration Roadmap (Estublier, 2000) reports several problems that need to be addressed when developing Configuration Management tools. SCCS-XP fully addresses some of the proposed issues since:

- It provides a flexible data model that can be used to represent complex data as well as data involving any user-defined relationship.
- It guarantees the interoperability with SE tools by supporting XML based data exchange. Scalability and efficiency are provided to the extent supported by the underlying database.
- It supports the definition of any relevant attribute as an XML tag; then appropriate queries can be used to extract and combine data according to the attributes' values.
- It provides mechanisms to extract the data and put them in the required format, place, etc. Actually this is done by means of the plug-in mechanism mentioned in Section 2.4.
- It is an extensible platform that can be enhanced in several respects. For instance, it is possible to plug-in the definition of new data models, to add support for impact analysis or for exchanging data with external tools, or for consistency check, and so on. SCCS-XP is actually open to incremental customization.

Other research efforts comply, to some extent, with the indications provided by the roadmap. Among the many initiatives, UVM (Westfechtel, Munch and Conradi, 2001) is probably one of the most conceptually advanced. UVM features uniform version model and support architecture for SCM. UVM provides a base model that can be employed to express specific version models. The underlying layered architecture is completely orthogonal to the data model used for representing software artifacts and their relationships. In UVM, version rules are placed at the bottom of the layered architecture and can be used for expressing different version models. With respect to UVM, our approach is more lightweight. SCCS-XP is not perfectly orthogonal with respect to the data model (which is XML), nevertheless, it employs XML as a meta language to define *Specific data models*. The result is that SCCS-XP is largely independent from the data model used for representing software artifacts.

Our approach shares some characteristics with NUCM (van der Hoek et al., 2002): both provide a quite generic repository, upon which several different SCM policies can be built. This is fundamental to free users form the responsibility of managing issues that can be dealt with at the platform level. MCCM (van der Lingen and van der Hoek, 2004) is a platform featuring a repository to which pluggable components can be added. MCCM, like NUCM, manages data at the level of files or file collections and features a distributed repository. On the contrary, SCCS-XP manages data at finer granularities, but does not manage distribution issues at all. Another relevant difference is that MCCM and NUCM feature a very powerful repository, so that a concrete SCM tool is obtained essentially by constraining the way data are managed by the repository itself. SCCS-XP allows plug-ins to extend the data model and the functionality provided by a lightweight repository.

Other systems, like Coven (Chu-Carroll and Sprenkle, 2000), address the granularity of artifacts. However, these are specialized systems that exploit document structuring for their very specific objectives. Coven manages fragments of source code corresponding to project elements in order to support collaboration in an environment integrating SCM and collaborative development.

EPOS (Conradi et al., 1995) is one of the best known process support environment. In order to effectively support the development process, EPOS consistently manages the software artifacts, thus featuring integrated process and product

management. The EPOS repository supports versioning and long transactions. The principles that inspired the creation of the EPOS repository are similar to the principle underlying SCCS-XP, in that both approaches deal with the management of complex structured artifacts, and tend to support a smooth and effective integration with the development environment. There are also important differences: EPOS uses a different versioning model (Change Oriented Versioning), and does not support fine-grained versioning. SCCS-XP does not provide native support for the software process, although in principle it would be possible to exploit the event-based architecture of SCCS-XP to build a process support plug-in.

Finally, the COOP/Orm project developed a collaborative SCM system integrated with a programming environment. COOP/Orm features a fine-grained versioning model (Magnusson and Asklund, 1996) similar to the one featured by SCCS-XP. COOP/Orm's approach to SCM suffers by some constraints: COOP/Orm comes with a built-in optimistic check-out mechanism with synchronous updates, it obliges the usage of the integrated editor, and it does not exchange data easily with external development tools.

5 CONCLUSIONS

SCCS-XP is a light-weight, XML-based platform providing basic SCM functionalities. SCCS-XP supports a data model which is actually a sort of meta-model that can be used to build support for several different types of software artifacts; integration, at the data level, with external tools is easy. A traditional, extensional versioning model is provided, but SCCS-XP provides the possibility to extend this model, as well as to introduce additional functionality. SCCS-XP was employed to realize prototypes of SCM tools ranging form a simple CVS-like tool to a sophisticated tool that can manage semantic relations, and perform different types of impact analysis.

In conclusion, we believe that SCCS-XP can be employed to build customized SCM environments suitable to support modern software development practices.

Future work will concern: 1) the development of a plug-in for UML, supporting the XMI data model; 2) the implementation of a locking mechanism 3) the implementation of a plug-in that allows the user to define new specific relations, and to define rules that can be triggered by specific events and invoke suitable actions; 4) the development of a plug-in that exploits the virtual files mechanism to supports different views of a single project.

ACKNOWLEDGEMENTS

We would like to thank Luca Ridolfi and Riccardo Serafin for their contribution in the design and implementation of the tool.

REFERENCES

Boag, S., Chamberlin, D., Fernández, M. F., Florescu, D., Robie, J., Siméon, J., 2007. *XQuery 1.0: An XML Query Language.* W3C Recommendation.

Chu-Carroll M.C. and Sprenkle S., 2002. Coven: Brewing Better Collaboration through Software Configuration Management. In *FSE 2000, San Diego*.

Clark, J., DeRose, S., 1999. *XML Path Language (XPath) Version 1.0*. W3C Recommendation.

Conradi R., Westfechtel B., 1998. Version Models for Software Configuration Management. In *ACM Computing Surveys*, Vol. 30, N. 2, pp. 232-282.

Conradi, R., Larsen, J., Nguyen, M.N., Munch, B.P., Westby, P.H., 1995. Integrated Product and Process Management in EPOS. *Journal of Integrated CAE*.

Estublier, J., 2000. Software configuration management: a roadmap. In *ICSE - Future of SE Track*.

Leblang, D. B., 1994. The CM challenge: Configuration management that works. In *Configuration Management*, Vol. 2 of Trends in Software, Wiley, pp. 1-37.

Magnusson, B., Asklund, U., 1996. Fine grained version control of configurations in COOP/Orm. In *ICSE 1996, SCM-6 Workshop*.

Mouat, A., 2002. *XML Diff and Patch Utilities.* CS4 Dissertation, Heriot-Watt University,.

Tichy, W. F., 1988. Tools for software configuration management. In *Proc. of the Int. Workshop on Software Version and Configuration Control*, Grassau, Germany, Teubner Verlag.

van der Hoek, A., Carzaniga, A., Heimbigner, D., Wolf, A. L., 2002. A Testbed for Configuration Management Policy Programming. *IEEE TSE*, Volume 28, Issue 1, pp. 79 - 99.

van der Lingen, R., van der Hoek, A., 2004. An Experimental, Pluggable Infrastructure for Modular Configuration Management Policy Composition. In *ICSE 2004*, Edinburgh.

Westfechtel B., Munch B. P., Conradi R., 2001. A Layered Architecture for Uniform Version Management. *IEEE TSE*. Volume 27, Issue 12.